Outline

1. LinkedLists to Binary Trees
2. Why Do We Care About Binary Trees?
3. Printing Recursively
4. Binary Tree Traversals

Back To LinkedLists

Consider the following standard LinkedList:

```
front
```

Recall the definition of a ListNode:

```
public class Node {
    public int data;
    public Node next;
}
```

What if we added more fields?
- Multiple data fields?
- Multiple "next" fields?

Introducing Trees

```
public class Node {
    public int data;
    public Node left;
    public Node right;
}
```

Nodes with Multiple next Fields

```
public class Node {
    public int data;
    public Node next1;
    public Node next2;
}
```

(white is right; yellow is left)

Binary Trees

```
root
```

```
public class Node {
    public int data;
    public Node left;
    public Node right;
}
```

(root)

(yellow is next2; red is next1)
Consider the following LinkedList of a mathematical expression:

\[(2 \times 4) + (7 - 3)\]

What's bad about it?
- It doesn't really help us with the structure
- Looking at it doesn't really show us what's going on

What about this structure instead?

```
+a
*b
-7
3
```

Now we can see the order of operations much more clearly!

More Uses of Trees
- Recursive Trees (including things like games of Tic-Tac-Toe)

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>
```

- Compression (this will be your last assignment!)

Uses of Trees
- Parsing (Programming Languages, Math, etc.)
- Implementing TreeSet
- Directory File Structure

Printing A LinkedList (Again)
```
1 public void print() {
2   Node current = this.front;
3   while (current != null) {
4     System.out.print(current.data + " ");
5     current = current.next;
6   }
7 }
```

We’d like to figure out how to print trees. Since LinkedLists are “simpler versions of trees”, they might help.

How do we go in every direction in a tree?

USE RECURSION!

Printing a LinkedList Recursively
```
To print a LinkedList...
- Print the front of the list
- Print the next of the list (recursively)
```

```
1 public void print() {
2   print(this.front);
3 }
```

```
5 public void print(Node c) {
6   if (c != null) {
7     System.out.print(c.data + " ");
8     print(c.next);
9   }
10 }
```

Printing a Tree Recursively
```
To print a BinaryTree...
- Print the root of the tree
- Print the left of the tree (recursively)
- Print the right of the tree (recursively)
```

```
1 public void print() {
2   print(this.root);
3 }
```

```
5 public void print(Node c) {
6   if (c != null) {
7     System.out.print(c.data + " ");
8     print(c.left);
9     print(c.right);
10 }
11 }
```
Printing a Tree Example

```java
public void print(Node c) { // c = 1
    if (c != null) {
        System.out.print(c.data + " ... + "");
        print(c.left);
        print(c.right);
    }
}
```

Trace
```
c
  1
  / \
2   3
  \ / \
 4   5
```

OUTPUT
```
>> 1 2 3
```
Printing a Tree Example

1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace

1 2 3 OUTPUT

Printing a Tree Example

1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace

1 2 3 OUTPUT

Printing a Tree Example

1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace

1 2 3 OUTPUT

Printing a Tree Example

1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace

1 2 3 OUTPUT

Printing a Tree Example

1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace

1 2 3 OUTPUT

Printing a Tree Example

1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace

1 2 3 OUTPUT

Printing a Tree Example

1 public void print(Node c) { // c = 1
2 if (c != null) {
3 System.out.print(c.data + " ");
4 print(c.left);
5 print(c.right);
6 }
7 }

Trace

1 2 3 OUTPUT
Printing a Tree Example

```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        if (c.left != null) {
            print(c.left);
        }
        print(c.right);
    }
}
```

Example 1:
```
c = new Node(1);
c.left = new Node(2);
c.right = new Node(3);
c.left.left = new Node(4);
c.left.right = new Node(5);
c.right.right = new Node(6);
```

Trace:
```
1 2 3 4 5 6
```

Example 2:
```
c = new Node(1);
c.left = new Node(2);
c.right = new Node(3);
c.right.left = new Node(4);
c.right.right = new Node(5);
```

Trace:
```
1 2 3 4 5 6
```

Example 3:
```
c = new Node(1);
c.left = new Node(2);
c.right = new Node(3);
c.right.right = new Node(5);
```

Trace:
```
1 2 3 5 6
```

---

Tree Traversals

Pre-Order Traversal
```
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

In-Order Traversal
```
public void print(Node c) {
    if (c != null) {
        print(c.left);
        System.out.print(c.data + " ");
        print(c.right);
    }
}
```

Post-Order Traversal
```
public void print(Node c) {
    if (c != null) {
        print(c.left);
        print(c.right);
        System.out.print(c.data + " ");
    }
}
```
Tree Traversal Example

Consider the following binary tree:

```
          10
         /   \
        22    3
       /     / \
      4     53  66
     /     /    \
    17    12    17
```

Compute the Pre-Order, In-Order, and Post-Order Traversals:

- **Pre-Order:** 10, 22, 3, 4, 53, 66, 17, 12
- **In-Order:** 3, 22, 4, 10, 66, 53, 17, 12
- **Post-Order:** 3, 4, 22, 66, 12, 17, 53, 10

Traversals Trick

To Quickly Generate A Traversal

- Trace a path around the tree
- As you pass a node on the proper side, process it:
  - **Pre-Order:** left
  - **In-Order:** bottom
  - **Post-Order:** right

Binary Tree method

Binary Tree methods are just normal recursive functions. The base case/recursive calls will always be similar.

Writing a Binary Tree Method

```
public type method(...) {
    return method(this.root, ...);
}
```

```
private type method(TreeNode current, ...) {
    if (current == null) {
        // DO BASE CASE
    } else if (current.data == value) {
        return true;
    } else {
        boolean leftContainsValue = contains(current.left, value);
        boolean rightContainsValue = contains(current.right, value);
        return leftContainsValue || rightContainsValue;
    }
}
```

Some Tree Tips!

- Trees are just generalized LinkedLists. So, all of the things you learned about references with LinkedLists are going to apply to trees as well.

- Almost all the tree methods you write will be recursive (and will have a private helper that takes in the root).

- Make sure you understand all the traversals; the trick can be very useful.

Binary Tree contains()

Write a method, in the IntTree class, called contains():

```
public boolean contains(int value) {
    return contains(this.root, value);
}
```

```
private boolean contains(IntTreeNode current, int value) {
    if (current == null) {
        return false;
    } else if (current.data == value) {
        return true;
    } else {
        boolean leftContainsValue = contains(current.left, value);
        boolean rightContainsValue = contains(current.right, value);
        return leftContainsValue || rightContainsValue;
    }
}
```